CS 175, Project in Artificial Intelligence

Lecture 4: Document Classification

Padhraic Smyth
Department of Computer Science
Bren School of Information and Computer Sciences
University of California, Irvine
Logistics

• Assignment 1: graded

• Assignment 2: Text Classification
  – Due on Thursday at noon
  – Will discuss in class today and again Monday

• Project Proposals
  – Will be due end of next week
  – Will discuss project ideas and proposals in more detail later this week

• Lectures:
  – Today: more on document classification, Assignment 2
  – Wed: more on projects and proposals
Assignment 1: Test Examples

# Problem 1
# test string (from Wikipedia)
s = 'The Declaration of Independence is the statement adopted by the Second Continental Congress meeting at the Pennsylvania State House (Independence Hall) in Philadelphia on July 4, 1776. It announced that the thirteen American colonies,[2] then at war with the Kingdom of Great Britain, regarded themselves as thirteen newly independent sovereign states, and no longer under British rule.'

print('Percent:', '%.2f' % letter_percentage(s,'t'))
print('Percent:', '%.2f' % letter_percentage(s,'e'))
print('Percent:', '%.2f' % letter_percentage(s,'z'))

# Correct solutions are
#  Percent: 10.83
#  Percent: 16.24
#  Percent: 0.00
Assignment 1: Test Examples for Problem 2

# Problem 2

testurl1 = 'http://www.ics.uci.edu/~smyth/courses/cs175'
testurl2 = 'http://www.ics.uci.edu/~smyth/courses/cs175/assignment2.html'

tokens = first_words(testurl1, 10)
print(tokens)
print()
tokens = first_words(testurl2)
print(tokens)

# Correct solutions

# ['CS', '175', '|', 'Winter', '2017', 'CS', '175', ':', 'Project', 'in']
# ['CS', '175', '|', 'Winter', '2017', 'CS', '175', ':', 'Assignment', '2', ',', 'Winter', '2007', 'Instructions', 'Reading', ':', 'Read', 'the', ....

...... 'before', 'you', 'submit', ']', 'Deadline', ':', 'Noon', 'on', 'Thursday', '26th']
Assignment 1: Test Examples for Problem 3

# Problem 3

s = 'The Declaration of Independence is the statement adopted by the Second Continental Congress meeting at the Pennsylvania State House (Independence Hall) in Philadelphia on July 4, 1776. It announced that the thirteen American colonies,[2] then at war with the Kingdom of Great Britain, regarded themselves as thirteen newly independent sovereign states, and no longer under British rule.'

z1, z2 = parts_of_speech(s)

# Correct solution:
# Total number of tokens is 68
# Tag: NOUN Percentage of tokens = 32.35
# Tag: ADP Percentage of tokens = 16.18
# Tag: . Percentage of tokens = 11.76
# Tag: ADJ Percentage of tokens = 10.29
# Tag: DET Percentage of tokens = 8.82
# Tag: VERB Percentage of tokens = 7.35
# Tag: ADV Percentage of tokens = 4.41
# Tag: NUM Percentage of tokens = 4.41
# Tag: PRON Percentage of tokens = 2.94
# Tag: CONJ Percentage of tokens = 1.47
Grading of Assignment 1

• 3 points for each of functions 1 and 2, 6 points for function 3

• Most students got full (12) points. All students in the range [9, 12]

• Typical point deductions (-1 for each)
  – No (or almost no) comments
  – Problem 1
    • Incorrect percentage calculations, e.g., do not sum to 100
    • Fractions instead of percentages
    • Character list contained non-alphabetical characters (e.g., numbers like 1, 7, 7, 6)
    • If a character is not in the string (e.g., no ‘z’ in test string s) the function should return 0.00% and not fail
  – Problem 3
    • Incorrect formatting (e.g., more than 2 decimal places)
    • Wrong tagset (did not use ‘universal’ tagset)
Assignment 2
Assignment 2: Document Classification

- Build and evaluate text classifiers using scikit-learn

- Extract text features: counts and TF-IDF

- Fit classifiers to training data and make predictions on new data

- Evaluate classifier accuracy

- Investigate which words have the most positive and most negative weight for each class

- Create a confusion matrix for a classifier given predictions and targets
Assignment 2: Update 1

Add the following at the start of your solution code in

(1) function extract_text_features
   # Replace FIRSTNAME_LASTNAME with your name
   print('Student: FIRSTNAME_LASTNAME, Function: extract_text_features()')

and

(2) function fit_and_predict_LR
   # Replace FIRSTNAME_LASTNAME with your name
   print('Student: FIRSTNAME_LASTNAME, Function: fit_and_predict_LR()')
Assignment 2: Update 2

Please add "shuffle=True, random_state=42" to the argument list when you call `fetch_20newsgroups()` on lines 60 and 61.

This should ensure that all students get the following numbers:

- Dimensions of $X_{\text{train\_counts}}$ are (2989, 39831)
- Number of non-zero elements in $X_{\text{train\_counts}}$: 372208
- Percentage of non-zero elements in $X_{\text{train\_counts}}$: 0.31
- Average number of word tokens per document: 181.08
- Average number of documents per word token: 13.59

```python
twenty_test.target_names
['rec.motorcycles', 'rec.sport.baseball',
 'rec.sport.hockey', 'sci.crypt', 'soc.religion.christian']```
Assignment 2: Update 3

Change the definition of

```python
def LR_weights_and_words(X_train, Y_train, M)
```

to

```python
def LR_weights_and_words(traindata, M):
```

where traindata is of type “bunch” in sklearn

(Will allow you to recover the names of the features with the highest and lower weights)
Working With Text Data

The goal of this guide is to explore some of the main scikit-learn tools on a single practical task: analysing a collection of text documents (newsgroups posts) on twenty different topics.

In this section we will see how to:

- load the file contents and the categories
- extract feature vectors suitable for machine learning
- train a linear model to perform categorization
- use a grid search strategy to find a good configuration of both the feature extraction components and the classifier

Tutorial setup

To get started with this tutorial, you firstly must have the scikit-learn and all of its required dependencies installed.

Please refer to the installation instructions page for more information and for per-system instructions.

The source of this tutorial can be found within your scikit-learn folder:

```
sklearn/doc/tutorial/text-analytics/
```

The tutorial folder, should contain the following folders:

- *.rst files - the source of the tutorial document written with sphinx
- data - folder to put the datasets used during the tutorial
- skeletons - sample incomplete scripts for the exercises
- solutions - solutions of the exercises

You can already copy the skeletons into a new folder somewhere on your hard-drive named sklearn_tut_workspace where you will edit your own files for the exercises while keeping the original skeletons intact:

```
% cp -r skeletons work_directory/sklearn_tut_workspace
```
Assignment 2

n_categories = 5
twenty_train, twenty_test = load_news_dataset(n_categories)

type(twenty_train)

   sklearn.datasets.base.Bunch

type(twenty_train.data) # the attribute data is a list of strings (docs)

   List

len(twenty_train.data) # number of strings (documents) is 2989

   2989

twenty_train.data[0] # contents of the first string (document)

    'From: Brian.Vaughan@um.cc.umich.edu (Brian Vaughan)
     Subject: For Sale: Kawasaki EX500 (Michigan)
     Organization: University of Michigan
     Lines: 13
     Distribution: world
     NNTP-Posting-Host: dssl.uis.umd.umich.edu
     * FOR SALE *
     From Ann Arbor, Michigan
     *1988 Kawasaki EX-500 \n6682 miles\nCherry Red\nExcellent condition\nAsking $2300\nContact Brian at (313) 747-1604 (days) \n(313) 434-7284 (evenings & weekends)\n or e-mail at vaughan@umich.edu...or reply to this post.'
Assignment 2

# create TFIDF representations of the training data and sample test data

X_train_counts, X_train_tfidf, X_test_counts, X_test_tfidf =
    extract_text_features(twenty_train.data, sample_test_documents, 1)

# the extract_text_features() function takes documents (as a list of strings) and returns bag of words as a sparse array of counts

# Uses CountVectorizer and TfidfTransformer

X_train_counts
<2989x39831 sparse matrix of type '<class 'numpy.int64'>'>
with 372208 stored elements in Compressed Sparse Row format>
CountVectorizer and TfidfTransformer

CountVectorizer
- Tokenizes sets of documents, creates and converts to bag of words

TfidfTransformer
- Converts bag of words into tfidf representation

The following link provides helpful examples and information:
http://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction
CountVectorizer Class

- **Input**
  - Set of documents in the form of a list of strings

- **Output**
  - Vocabulary (after tokenization)
  - Bag of words (counts) in sparse array format

```python
CountVectorizer(analyzer='word', binary=False,
    decode_error='strict', dtype='numpy.int64',
    encoding='utf-8', input='content', lowercase=True,
    max_df=1.0, max_features=None, min_df=1,
    ngram_range=(1, 1), preprocessor=None, stop_words=None,
    strip_accents=None, token_pattern='(?u)\b\w\w+\b',
    tokenizer=None, vocabulary=None)
```
Example of CountVectorizer Usage

```python
>>> vectorizer = CountVectorizer(min_df=1) # create an instance

>>> corpus = [ ... 'This is the first document.', ... 'This is the second second document.', ... 'And the third one.', ... 'Is this the first document?', ... ] # define a sample set of documents

>>> X = vectorizer.fit_transform(corpus) # apply fit_transform method

>>> X
<4x9 sparse matrix of type '<... 'numpy.int64'>>' with 19 stored elements in Compressed Sparse ... format>
```

From http://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction
Example of CountVectorizer Usage

From http://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction

```python
>>> vectorizer = CountVectorizer(min_df=1)  # create an instance

>>> corpus = [ ... 'This is the first document.', ... 'This is the second second document.', ... 'And the third one.', ... 'Is this the first document?', ... ]  # define a sample set of documents

>>> X = vectorizer.fit_transform(corpus)  # apply fit_transform method
>>> X
<4x9 sparse matrix of type '<... 'numpy.int64'>>' with 19 stored elements in Compressed Sparse ... format>

>>> names = vectorizer.get_feature_names()  # recover feature names
>>> names
['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this']

>>> vectorizer.vocabulary_.get('second')  # index of term using the vocabulary attribute
5
```
### CountVectorizer Methods


<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>build_analyzer()</td>
<td>Return a callable that handles preprocessing and tokenization</td>
</tr>
<tr>
<td>build_preprocessor()</td>
<td>Return a function to preprocess the text before tokenization</td>
</tr>
<tr>
<td>build_tokenizer()</td>
<td>Return a function that splits a string into a sequence of tokens</td>
</tr>
<tr>
<td>decode (doc)</td>
<td>Decode the input into a string of unicode symbols</td>
</tr>
<tr>
<td>fit (raw_documents[, y])</td>
<td>Learn a vocabulary dictionary of all tokens in the raw documents.</td>
</tr>
<tr>
<td>fit_transform (raw_documents[, y])</td>
<td>Learn the vocabulary dictionary and return term-document matrix.</td>
</tr>
<tr>
<td>get_feature_names ()</td>
<td>Array mapping from feature integer indices to feature name</td>
</tr>
<tr>
<td>get_params ([deep])</td>
<td>Get parameters for this estimator.</td>
</tr>
<tr>
<td>get_stop_words ()</td>
<td>Build or fetch the effective stop words list</td>
</tr>
<tr>
<td>inverse_transform (X)</td>
<td>Return terms per document with nonzero entries in X.</td>
</tr>
<tr>
<td>set_params (**params)</td>
<td>Set the parameters of this estimator.</td>
</tr>
<tr>
<td>transform (raw_documents)</td>
<td>Transform documents to document-term matrix.</td>
</tr>
</tbody>
</table>
Example of CountVectorizer Usage

```python
>>> X
<4x9 sparse matrix of type '<... 'numpy.int64'>' with 19 stored elements in Compressed Sparse ... format>

>>> X.toarray()
array([[0, 1, 1, 1, 0, 0, 1, 0, 1],
       [0, 1, 0, 1, 0, 2, 1, 0, 1],
       [1, 0, 0, 1, 0, 1, 1, 0],
       [0, 1, 1, 1, 0, 0, 1, 0, 1]])

>>> vectorizer.transform(['Something completely new.']).toarray()
... array([[0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0]])
```

Also:
- Can use CountVectorizer to generate bigrams (-> much larger vocabulary)
TfidfTransformer Class

- Transforms a count matrix to a tf-idf representation

- Various parameters: use default settings in Assignment 2

```python
from sklearn.feature_extraction.text import TfidfTransformer

tfidf_transformer = TfidfTransformer()

# fit/compute Tfidf weights using "data 1"
data1_tfidf = tfidf_transformer.fit_transform(data1)

# apply fitted weights to "data2"
data2_tfidf = tfidf_transformer.transform(data2)
```
TfidfTransformer Class

From http://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction
Linear Classifiers
Linear Classifiers for 2-Class Problems

A linear classifier computes a linear weighted sum of the inputs

e.g., in 2 dimensions, with 2 classes

\[ f = \text{classifier output} = w_0 + w_1 x_1 + w_2 x_2 \]

Why do we need this extra constant weight?
Geometric Interpretation of a Linear Classifier

Note that the decision boundary corresponds to the points where

\[ f = 0, \text{ i.e., } w_0 + w_1 x_1 + w_2 x_2 = 0 \]

which is the equation of a line in 2 dimensions.

The \( w_0 \) weight allows us to have lines that have non-zero intercept, i.e., that don’t need to go through the origin.
Linear Classifier with Overlapping Class Distributions

TWO-CLASS DATA IN A TWO-DIMENSIONAL FEATURE SPACE
A Linear Classifier (with 2 Features)

Inputs

\[ f(x_1, x_2) = w_0 + w_1 x_1 + w_2 x_2 \]

Threshold Function

Output = class prediction

\[ f > 0? \]

1 or 2
Linear Classifiers for 2-Class Problems

A linear classifier computes a linear weighted sum of the inputs

e.g., in 2 dimensions
\[ f(x_1, x_2) = w_0 + w_1 x_1 + w_2 x_2 \]

and more generally in T dimensions
\[ f(x) = f(x_1, ..., x_T) = \sum_j w_j x_j = w_0 + w_1 x_1 + w_2 x_2 + ... + w_T x_T \]
Linear Classifiers for 2-Class Problems

A linear classifier computes a **linear weighted sum** of the inputs

e.g., in 2 dimensions

\[ f(x_1, x_2) = w_0 + w_1 x_1 + w_2 x_2 \]

and more generally in T dimensions

\[ f(x) = f(x_1, \ldots, x_T) = \sum_j w_j x_j = w_0 + w_1 x_1 + \ldots + w_T x_T \]

**Sidenote:** this can also be written as the **inner product** of a weight-vector and the feature vector,

i.e., \[ f(x) = \sum_j w_j x_j = w^T x, \]

where \( w = (w_0, w_1, \ldots, w_T) \) and \( w^T \) is the transpose of \( w \)
Linear Classifiers for Text Documents

Linear classifiers use a weighted sum of the inputs
– With T features we have T + 1 weights (one per feature plus one “intercept”)

Examples of Linear Classifiers
– Linear Classifier (Perceptron)
– Logistic Regression <- widely used in practice, is what we will focus on
– Naïve Bayes (not so obvious, but its linear)
– Support Vector Machines

Example of a Non-Linear Classifier
– Neural Networks <- will be reviewed in future lectures

For further discussion of linear and non-linear classifiers see:
## Possible Weights for a Linear Classifier with Documents

<table>
<thead>
<tr>
<th>Class Label</th>
<th>predict</th>
<th>finance</th>
<th>stocks</th>
<th>goal</th>
<th>score</th>
<th>team</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>
Getting Class Probabilities with the Logistic Regression Model
Getting Class Probabilities....

Estimates of class probabilities $P(c \mid x)$ are very useful in practice, e.g., for ranking documents to show to a human user.
Getting Class Probabilities....

Estimates of class probabilities $P(c \mid x)$ are very useful in practice e.g., for ranking documents to show to a human user

Assume for simplicity we have a 2-class binary classification problem

Say we tried to get a probability of a class with a linear model:

$$P(c \mid x) = f(x) = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_T x_T$$

There is a problem:

$f(x)$ could be negative, could be $>1$, etc.
A Better Approach

\[ P(c \mid x) = f(x) = g(w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_T x_T) \]

where \( g(z) = \frac{1}{1 + e^{-z}} \)

As \( z \to \) positive infinity, \( g(z) \to 1 \), \( P(\text{class}) \to 1 \)
As \( z \to \) negative infinity, \( g(z) \to 0 \), \( P(\text{class}) \to 0 \)

This is the logistic regression model

In effect: a linear (weighted sum) model where the sum is transformed to lie between 0 and 1

...and we can interpret \( f(x) \) directly as a probability between 0 and 1
What does the Logistic Function look like?

\[ f(x) = g(z) = \frac{1}{1 + e^{-z}} \]

As \( z \to \) positive infinity, \( g(z) \to 1, \quad P(\text{class}) \to 1 \)

As \( z \to \) negative infinity, \( g(z) \to 0, \quad P(\text{class}) \to 0 \)

\[ z = \text{weighted sum} = \sum_{j} w_j x_j \]
Logistic Regression as a Neural Network

- Logistic regression can be viewed as a simple “artificial neuron”

Each “edge” in the network has an associated weight or parameter, $w_j$

$$f(x) = \frac{1}{1 + e^{-\sum_{j=0}^{T} w_j x_j}}$$
A Neural Network with 1 Hidden Layer

Here the model learns 3 different logistic functions, each one a “hidden unit” and then combines the outputs of the 3 to make a prediction.

This model is representationally more powerful than a single logistic function, but has many more parameters (can overfit unless we are careful).

The model can be trained using gradient methods – but local minima are a problem.
Deep Learning: Models with 2 or More Hidden Layers

We can build on this idea to create “deep models” with many hidden layers.

The model $f(x)$ is now a very flexible highly non-linear function.

Significant current interest in “deep learning” (e.g., 5, 10, 20 layers)
Explaining Decisions by an AI Algorithm
Explaining an Algorithm’s Decisions

• Generating human-interpretable explanations of decisions made by AI systems is very important to human users of these systems, e.g., in
  – Autonomous driving
  – Medical diagnosis
  – Product recommendations
  – And so on.....

• For linear classifiers, where we have 1 weight per input, this is straightforward
  – For each class, look at most positive weights and most negative weights
    • This tells us which features/terms (if present) have the most impact (Assignment 2)
    • For documents note that some terms might be rare: so we could measure how much impact they have on average, rather than when they are present
    • Can also tell the user which terms in a particular document contributed most to a decision

• For non-linear classifiers (such as neural networks), explaining decisions is much more complicated to do
ELI5

ELI5 is a Python package which helps to debug machine learning classifiers and explain their predictions.

hi there, i am here looking for some help. my friend is a interic graphics software on pc. any suggestion on which software tc sophisticated software (the more features it has, the better)

It provides support for the following machine learning frameworks and packages:

- **scikit-learn.** Currently ELI5 allows to explain weights and predictions of scikit-learn linear classifiers and regressors, print decision trees as text or as SVG, show feature importances and explain predictions of decision trees and tree-based ensembles. ELI5 understands text processing utilities from scikit-learn and can highlight text data accordingly. It also allows to debug scikit-learn pipelines which contain HashingVectorizer, by undoing hashing.
- **xgboost** - show feature importances and explain predictions of XGBClassifier and XGBRegressor.
- **lightning** - explain weights and predictions of lightning classifiers and regressors.
- **sklearn-crfsuite.** ELI5 allows to check weights of sklearn_crfsuite.CRF models.

From: https://github.com/TeamHG-Memex/eli5
This starts to make more sense. Columns are target classes. In each column there are features and their weights. Intercept (bias) feature is shown as `<BIAS>` in the same table. We can inspect features and weights because we're using a bag-of-words vectorizer and a linear classifier (so there is a direct mapping between individual words and classifier coefficients). For other classifiers features can be harder to inspect.

from: brian@ucsd.edu (brian kantor) subject: re: help for kidney stones ............... organization: the avant-garde of the now, ltd. lines: 12 nntp-posting-host: ucsd.edu as i recall from my bout with kidney stones, there isn’t any medication that can do anything about them except relieve the pain. either they pass, or they have to be broken up with sound, or they have to be extracted surgically. when i was in, the x-ray tech happened to mention that she’d had kidney stones and children, and the childbirth hurt less. demerol worked, although i nearly got arrested on my way home when i barfed all over the police car parked just outside the er. - brian
y=soc.religion.christian (probability 0.001, score -7.157) top features

<table>
<thead>
<tr>
<th>Contribution</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.258</td>
<td>&lt;BIAS&gt;</td>
</tr>
<tr>
<td>-6.899</td>
<td>Highlighted in text (sum)</td>
</tr>
</tbody>
</table>


as i recall from my bout with kidney stones, there isn't any medication that can do anything about them except relieve the pain. either they pass, or they have to be broken up with sound, or they have to be extracted surgically. when i was in, the x-ray tech happened to mention that she'd had kidney stones and children, and the childbirth hurt less. demerol worked, although i nearly got arrested on my way home when i barfed all over the police car parked just outside the er. - brian
Example (in Python) of Classifying Yelp Reviews

(code from Dimitris Kotzias, PhD student, Computer Science Department, UCI)
Yelp Dataset Challenge

Yelp Dataset Challenge rides again! Round 6 is here.
We’ve had 5 rounds, over $35,000 in cash prizes awarded, hundreds of academic papers written, and we are excited to see round 6.

Our dataset for this iteration of the challenge is the same as the last iteration - we’re sure there are plenty of interesting insights still waiting there for you. If you want the latest check-ins and reviews, don’t worry, we’ll have them for you in 2016 (along with some new attributes if you’re good). This set includes information about local businesses in 10 cities across 4 countries. This treasure trove of local business data is waiting to be mined and we can’t wait to see you push the frontiers of data science research with our data.

The Challenge Dataset:
- 1.6M reviews and 500K tips by 366K users for 61K businesses
- 481K business attributes, e.g., hours, parking availability, ambience.
- Social network of 366K users for a total of 2.9M social edges.
- Aggregated check-ins over time for each of the 61K businesses

Get the Data

Cities:
- U.K.: Edinburgh
- Germany: Karlsruhe
- Canada: Montreal and Waterloo
- U.S.: Pittsburgh, Charlotte, Urbana-Champaign, Phoenix, Las Vegas, Madison
Real Example from Yelp Data

Simple pipeline for classification of Yelp Reviews

- Extract the restaurant reviews
- Convert them to a tf*idf array
- Split data into training and testing
- Train on training data, and Test

```python
if __name__ == '__main__':
    extract_restaurant_reviews()
    X, Y = convert_to_array()
    X_train, X_test, Y_train, Y_test = split_data(X, Y)
    train_and_test(X_train, X_test, Y_train, Y_test)
```
Real Example from Yelp Data

<table>
<thead>
<tr>
<th>Yelp Dataset</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Reviews</td>
<td>706,693</td>
</tr>
<tr>
<td>Number of Reviews w/o Neutral Rating</td>
<td>595,468</td>
</tr>
<tr>
<td>Number of Tokens</td>
<td>85,392,376</td>
</tr>
<tr>
<td>Vocabulary Size w/o Stopwords</td>
<td>176,114</td>
</tr>
<tr>
<td>Array Dimensions</td>
<td>(595468, 176114)</td>
</tr>
<tr>
<td>Number of cells in the Array</td>
<td>104,870,251,352</td>
</tr>
<tr>
<td>Non-zero entries</td>
<td>28,357,001</td>
</tr>
<tr>
<td>Density</td>
<td>0.000027027</td>
</tr>
</tbody>
</table>
Histogram of Review Lengths
Real Example from Yelp Data

```python
def extract_restaurant_reviews():
    # get all the ids of restaurants
    ids = set()
    with open('./data/yelp/restaurants.json', 'r') as jfile:
        for line in jfile:
            data_point = json.loads(line)
            ids.add(data_point['business_id'])
    print('Total restaurants: ', len(ids))

    # get all the reviews
    reviews = []
    with open('./data/yelp/yelp_academic_dataset_review.json', 'r') as jfile:
        for line in jfile:
            r = json.loads(line)
            id = r['business_id']  # if business is a restaurant
            if id in ids:
                reviews.append(r)

    # save the reviews
    with open('./data/yelp/restaurant_reviews.json', 'w') as output_file:
        json.dump(reviews, output_file)
        output_file.write("\n")
    print('A total of ', len(reviews), ' reviews')

# Number of restaurants: 14,308
# A total of 706,693 reviews
```
# Real Example from Yelp Data

```python
# Grab only the text, then convert it to a tf*idf matrix

def convert_to_array(min_pos=4, max_neg=2):
    dir = './data/yelp/
    name = dir + 'restaurant_reviews.json'  # load data
    with open(name, 'r') as jfile:
        data = json.load(jfile)

    text = []
    Y = []
    for d in data:  # keep only the text and label
        review = d['text']
        stars = int(d['stars'])
        if stars >= min_pos:  # translate number of stars to binary
            score = 1
        elif stars <= max_neg:
            score = 0
        else:
            continue  # do not consider neutral

        text.append(BeautifulSoup(review).get_text())
        Y.append(score)

    # parameters should change depending on problem
    vectorizer = TfidfVectorizer(stop_words='english', max_df=1.0, min_df=0.0)  # this is awe
    X = vectorizer.fit_transform(text)

    print 'data shape: ', X.shape
    return X, Y

data shape: (595468, 176114)
```
Real Example from Yelp Data

```python
# split to train and test
def split_data(X, Y, test_size=0.5):
data_train, data_test, labels_train, labels_test = train_test_split(X, Y, test_size=test_size, random_state=42)
# important to be random, but have same results across different runs ;

print 'training size: ', data_train.shape[0],
print 'testing size: ', data_test.shape[0]  # careful these are sparse matrices

return data_train, data_test, labels_train, labels_test
```

training size: 297734
testing size: 297734
def train_and_test(X_train, X_test, Y_train, Y_test):

    # Specify the model. Again parameters should change
    logreg = linear_model.LogisticRegression(penalty='l2', fit_intercept=True)  # fit_intercept= bias

    # Train....
    logreg.fit(X_train, Y_train)
    pickle.dump(logreg, open('./data/yelp.log_model.pkl', 'w'))  # save in case we need later

    print 'Training: ',
    predicted = logreg.predict(X_train)  # Test
    print 'acc:', metrics.accuracy_score(Y_train, predicted)

    print 'Testing: ',
    predicted = logreg.predict(X_test)  # Test
    probs = logreg.predict_proba(X_test)
    print 'acc:', metrics.accuracy_score(Y_test, predicted)
    print 'auc:', metrics.roc_auc_score(Y_test, probs[:, 1])  # this is easy to plot as well

Training:  acc: 0.95586
Testing:  acc: 0.94812
auc: 0.98233
Overall takes about 15-20 mins to run (may produce some warnings)
Other Aspects of Document Classification
Examples of Labels/Categories/Classes

- Labels for documents or web-pages
  - Labels are often general categories
  - e.g., for news articles
    - "finance," "sports," "news>world>asia>business"
  - e.g., for biomedical articles
    - “gene expression”, “microarray”, “lung cancer”

- Labels may be genres
  - "editorials" "movie-reviews" "news"

- Labels may be opinion on a person/product
  - “like”, “hate”, “neutral”

- Labels may be domain-specific
  - "interesting-to-me" : "not-interesting-to-me"
  - “contains adult language” : “doesn’t”
  - language identification: English, French, Chinese, ...
  - “link spam” : “not link spam”
Where do Document Labels come from?

- Manually assigned (expensive)
  - Predefined dictionary of labels
  - Human labelers read all or part of the article and assigning the most likely label
  - Who are the labelers?
    - Domain experts
    - Librarians/editors (e.g., for the New York Times)
    - Low-paid labelers, e.g., via Amazon Turk
  - This is a subjective process
    - Even domain experts will disagree on some labels
    - In many cases there is no absolute “right” or “wrong” labeling

- Semi-automated process
  - e.g., domain experts define selected keywords for each label
  - Keyword matching used to return documents with most keyword matches for each label
  - Experts then label these returned documents
  - Classifier trained on these labeled documents
Other Aspects of Document Labels

• Large numbers of label values
  – Many applications have a very large number of possible class labels (thousands)
  – Distribution of labels is often highly skewed
    • Some labels very common, other labels very rare

• Multi-Label versus Single-Label documents
  – Multi-Label: each document can have multiple labels
  – Single-Label: each document is assigned a single label
  – The multi-label problem is more complex to handle
    • E.g., the model needs to decide how many labels to assign to each document
      (we will assume single-label for now, return to multi-label later)

• Hierarchical labels
  – Common in real-world applications that labels are related hierarchically in a tree
    • e.g., "news>world>asia>business"
  – Classifiers that use this hierarchy will generally perform better than classifiers that ignore it
Feature Selection

• Performance of text classification algorithms can often be improved by selecting only a subset of the terms

• Greedy search
  – Start from empty set or full set and add/delete one at a time
  – Heuristics for adding/deleting
    • Information gain (mutual information of term with class)
    • Chi-square
    • Other ideas

  – Methods tend not to be particularly sensitive to the specific heuristic used for feature selection, but some form of feature selection often improves performance
Feature Selection using Mutual Information

Average mutual information between (a) C, the class label and (b) \( f_t \), the presence or absence of a term in a document, defined as

\[
I(C; W_t) = H(C) - H(C|W_t) \\
= -\sum_{c\in C} P(c) \log(P(c)) \\
+ \sum_{f_t\in\{0,1\}} P(f_t) \sum_{c\in C} P(c|f_t) \log(P(c|f_t)) \\
= \sum_{c\in C} \sum_{f_t\in\{0,1\}} P(c, f_t) \log \left( \frac{P(c, f_t)}{P(c)P(f_t)} \right),
\]

From McCallum and Nigam, 1998

Where here c is the class and \( f_t \) indicates the presence or absence of term t

Typical approach: compute for all terms, include the top K terms in the classifier, and optimize the value of K via cross-validation (next lecture)
Generating Multi-Word Terms

- Consider multi-word terms like “New York”
  - Would rather treat this as one word “New York” rather than “New” and “York”

- We can extend our vocabulary to include multi-word terms (or ngrams)
  - Ngrams with n=1,2,3,4… e.g., “University of California Irvine” (n=4)

- Finding candidate n-grams
  - Space of possible multi-word combinations is huge
  - W word tokens: $W^2$ bigrams, $W^3$ trigrams, etc. (W order of $10^5$)
  - General approach: select ngrams that occur frequently
    - Keep track of all k-frequent ngrams in the corpus (e.g., k=10)
    - Use feature selection (e.g., mutual information) to select best
  - Can also use other filters to find good terms,
    - e.g., use a parser to automatically extract noun-phrases
      The big dog jumped over the lazy brown cat
Next Lecture

- Questions about Assignment 2
- Discussion of types of class projects
- Discussion about submitting project proposals
Backup Slides (from last lecture)
Background Reading: useful for Project Ideas

- Class Web site: http://www.ics.uci.edu/~smyth/courses/cs175
  - Additional links on Software and Demos for Text Analysis
  - Additional links for Data Sets

- 4 excellent online reference books
  - Speech and Language Processing, https://web.stanford.edu/~jurafsky/slp3/
  - A Course in Machine Learning, http://ciml.info/
Example of a Pipeline for Document Classification

Training Documents (corpus)

Tokenization → Lists of Tokens → Bag of Words → Machine Learning Algorithm

Stopword and rare word removal → Vocabulary → Frequency Counts

Document Classifier
Example of a Pipeline for Document Classification

1. **Training Documents (corpus)**
   - Tokenization
   - Lists of Tokens
   - Stopword and rare word removal
   - Vocabulary
   - Bag of Words
   - Frequency Counts
   - Machine Learning Algorithm

2. **New Document**
   - Tokenization
   - Lists of Tokens
   - Bag of Words
   - Document Classifier
   - Label Prediction
TF-IDF Weighting of Features

In practice the inputs can be weighted

- It can be helpful to use “TF-IDF weights” instead of counts

\[
\text{TF}(t,d) = \text{term frequency}, \text{i.e., number of times term } t \text{ occurs in doc } d
\]

\[
\text{IDF}(t,d) = \text{inverse document frequency}
= \log \left( \frac{N}{\text{number of docs with term } t} \right)
= \log \left( \frac{N}{\text{number of docs with term } t} \right)
\]

\[
\text{TF-IDF}(t,d) = \text{TF}(t,d) \times \text{IDF}(t,d)
\]

The IDF term has the effect of **upweighting terms that occur in few docs**
TF-IDF Example

N = 1000 in a corpus of news articles

Term 1: t = “city”, appears in 500 documents

\[ \text{IDF}(t) = \log(1000/500) = \log(2) = 1 \]  
(log is base 2, not important)

Term 2: t = “freeway”, appears in 10 documents

\[ \text{IDF}(t) = \log(1000/10) = \log(100) = 6.64 \]

So occurrences of “freeway” will get upweighted by a factor of 6.64 compared to occurrences of “city”